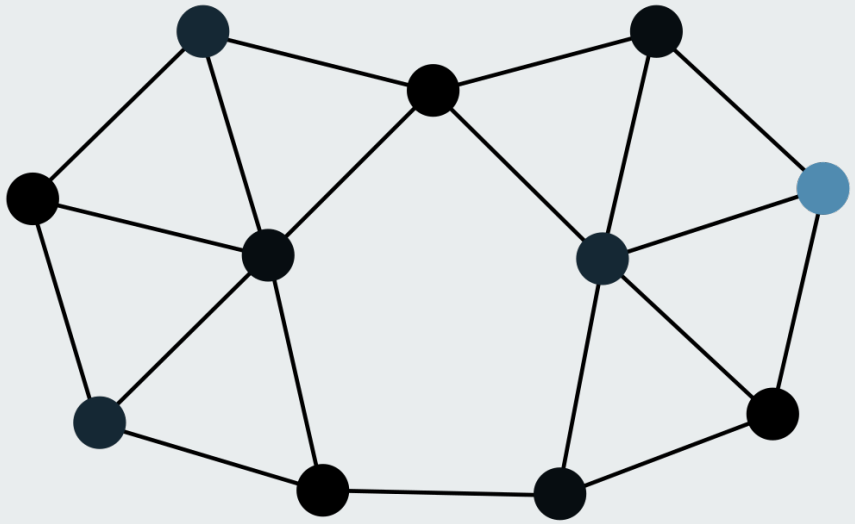
Distributed Systems

Architectures



Distributed Systems Architectures

Distributed systems are composed of multiple machines so organisation is important.

There is both a physical and a logical organisation.

The logical architecture may not follow the physical one.

Systems and Software Architectures

Software architectures: tell us how the various software components should be organised

- Also details their interactions

System architectures: tells us where the software components should be placed on physical machines and the physical links between nodes

There are many different variations of both architectures

Software Architecture

Architectural Style

Details the components, their connections to each other, the data exchanged between them, and how the components are configured into a single system

Component: a modular unit with well-defined interfaces

Connection: a mechanism for mediating communications between components

There are **four main architectures**…

- Layered
- Object-based
- Data Centred
- Event Based

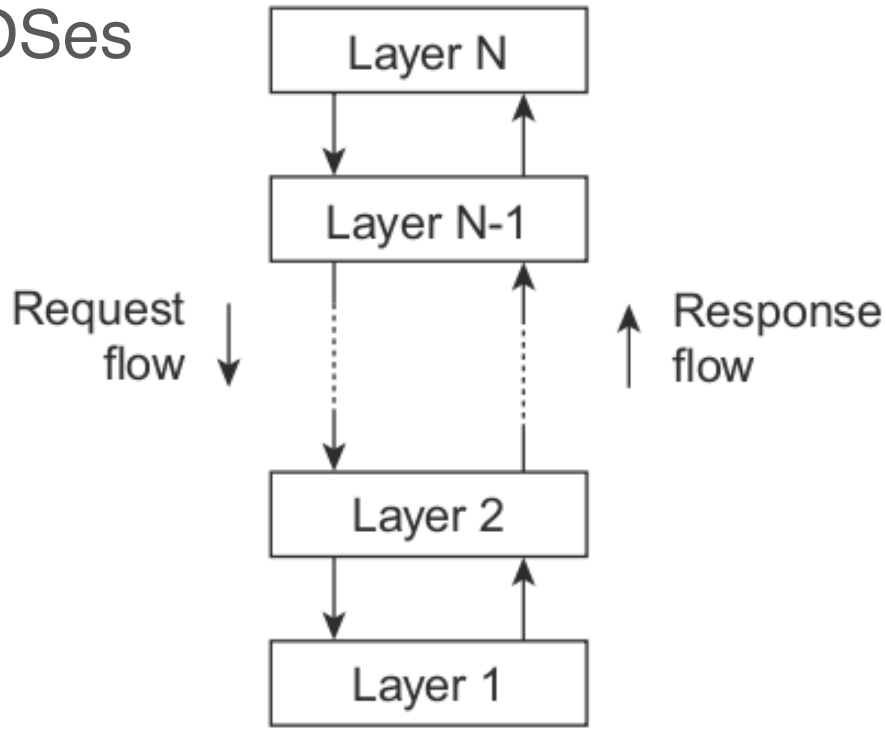
Layered Architecture

Components are organised into layers: a component at L_n can call components at L_{n-1} but not to L_{n+1}

Adopted by most networking and OSes

Requests flow top-to-bottom. ↓

Results flow bottom up. ↑

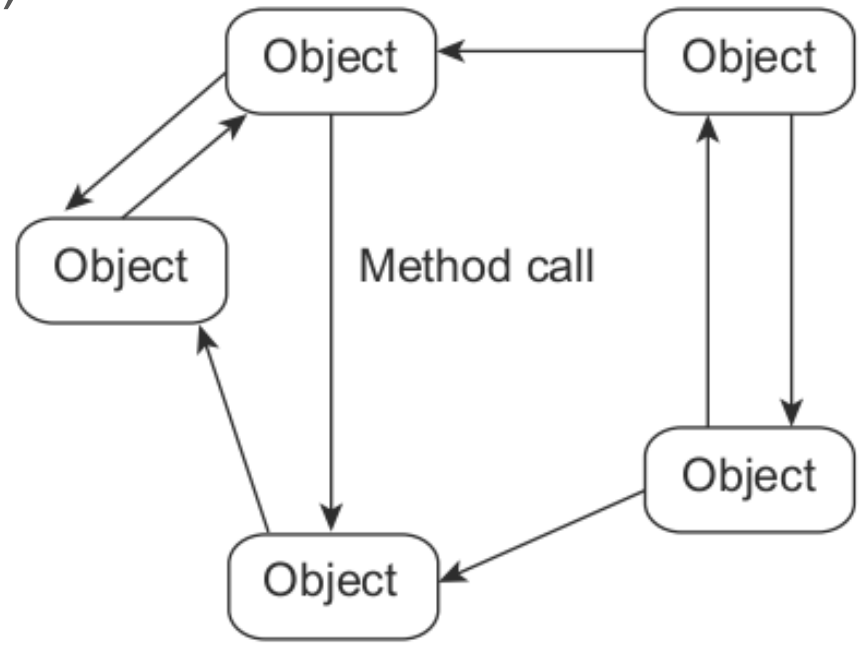


Object Based Architecture

Much looser organisation of components

Each component is defined by an object, the connections are remote procedure calls (RPC)

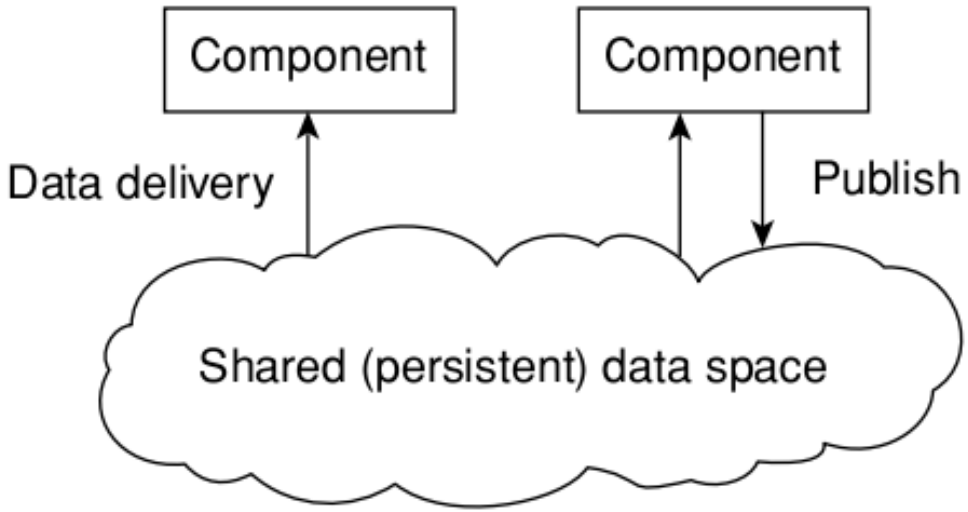
Similar in style to client server architectures



Data Centred Architecture

Processes communicate through a **common data repository**.

Examples of this include applications that share information through distributed file systems (Hadoop, Map-Reduce etc.)

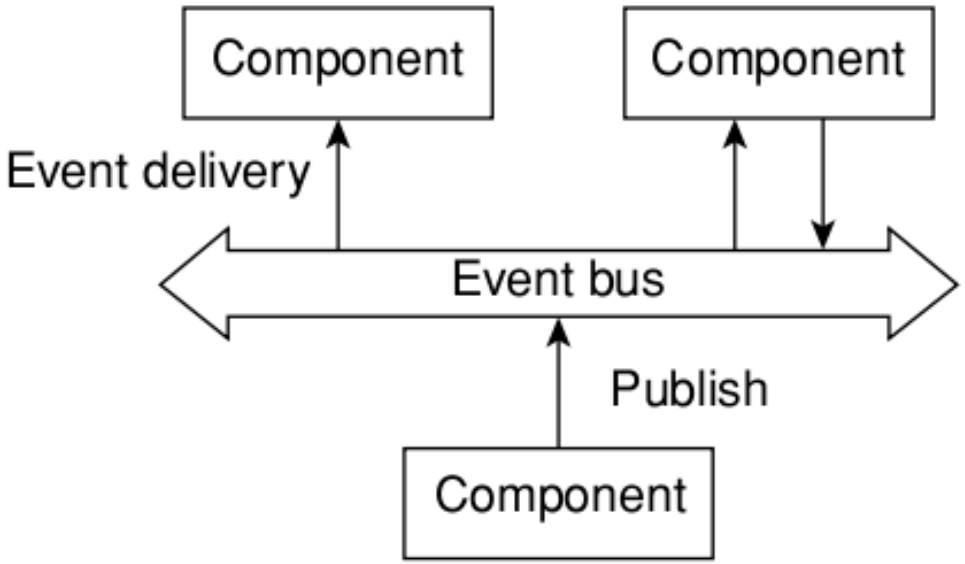


Event Based Architecture

Communication through events.

Example: Publish/subscribe systems

One or more process publish events, subscribed processes receive events and react.



System Architecture

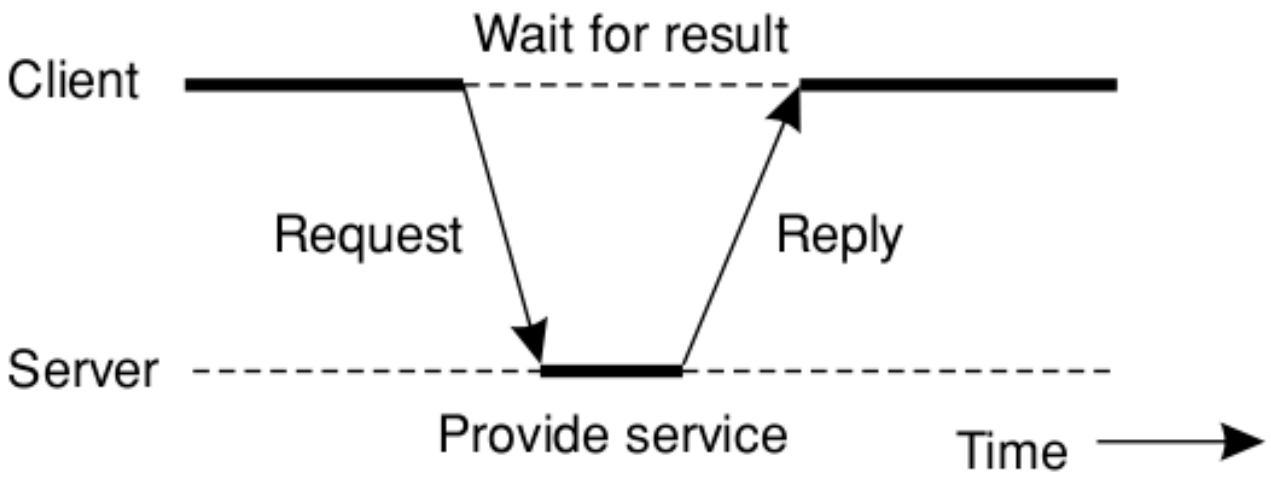
Centralised Systems

Client/Server

Helps manage the complexity of distributed systems

Components divided into groups of clients and servers (may overlap).

Clients request a service from the server. (request-reply behaviour)



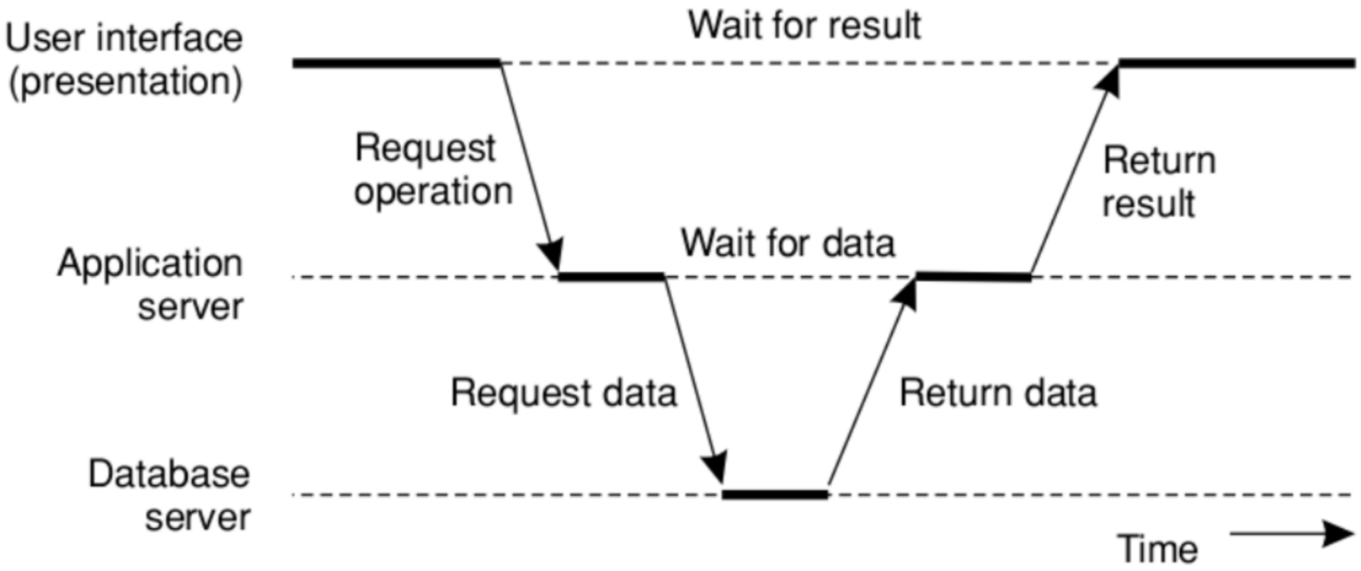
Multi-Tiered Architecture

Two or more layers, and processing is distributed among them.

A **website** for example.

Client renders a UI and makes requests to the web server

Server does some processing (might make further requests to a database).



Decentralised / Peer-To-Peer Systems

Decentralised Architecture

Centralised multi-tier architectures use vertical distribution where components are separated into different layers.

But it is also possible to split up servers and clients into many different parts and distribute processing that way, to get horizontal distribution.

However if we go a step further and make every node both a client and a server we end up with a **decentralised** or **peer-to-peer system**

Overlay Network

How can we organise nodes in a P2P system?

An overlay network is **a network built on top of another network**.

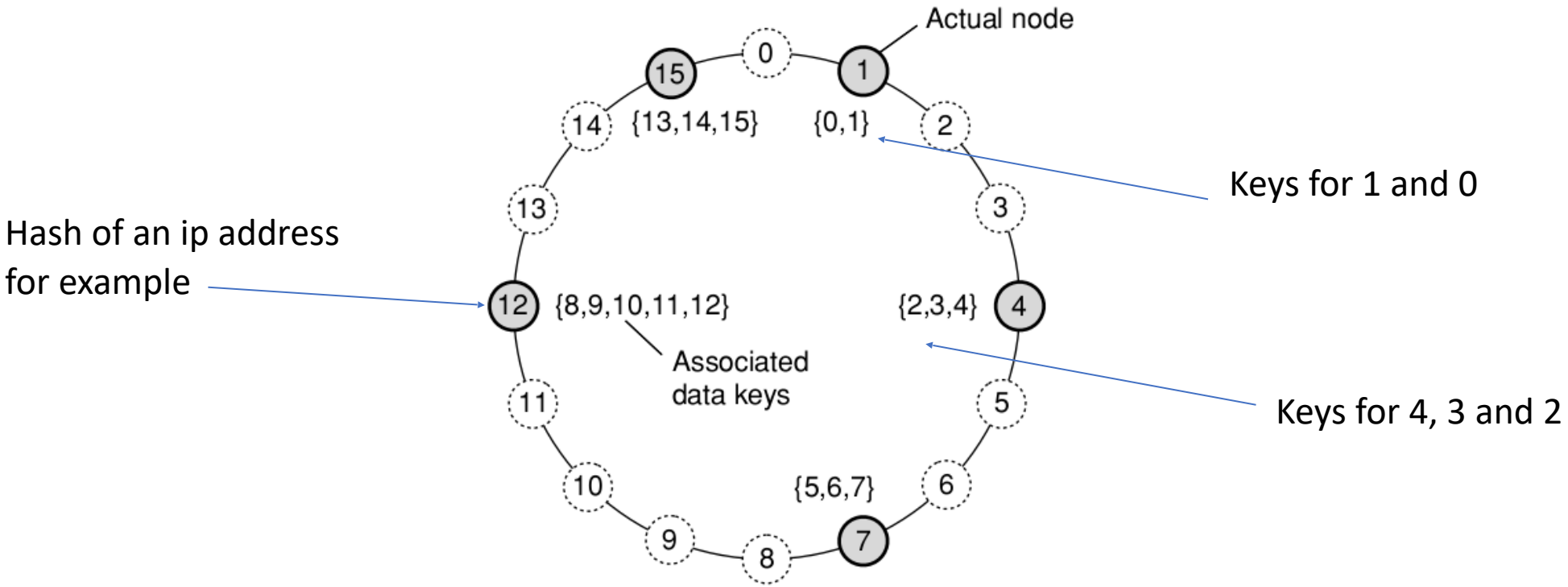
- Two processes cannot communicate directly
- Must send messages along their communication channels

Structured Peer To Peer Networks

Overlay Network using distributed hash table (DHT) - **more about this later**

Nodes are assigned random IDs (Hashed using a 160+ bit space - unlikely to collide)

An efficient and deterministic method to uniquely map the key of the item to the identifier of the node (using a distance metric perhaps)



Structured Peer To Peer Networks

Nodes are required to organise themselves into this overlay network through a process of membership management

Each node maintains links to other nodes such that lookups are done in *log n* time.

When a node wishes to join it generates an id and performs a lookup to find a successor node.

The node links to the successor (and predecessor) and inserts itself into the table.

Unstructured P2P Networks

- Uses randomised algorithms when generating the overlay network.
- Each node maintains a list of neighbours (and has a **different set** of neighbours).
- The goal of each node in an unstructured network is to generate an overlay that looks similar to a random graph
- All nodes maintain a list of *c* neighbours
- Each node represents a randomly chosen live node.
- Impossible to get exact same list on each join

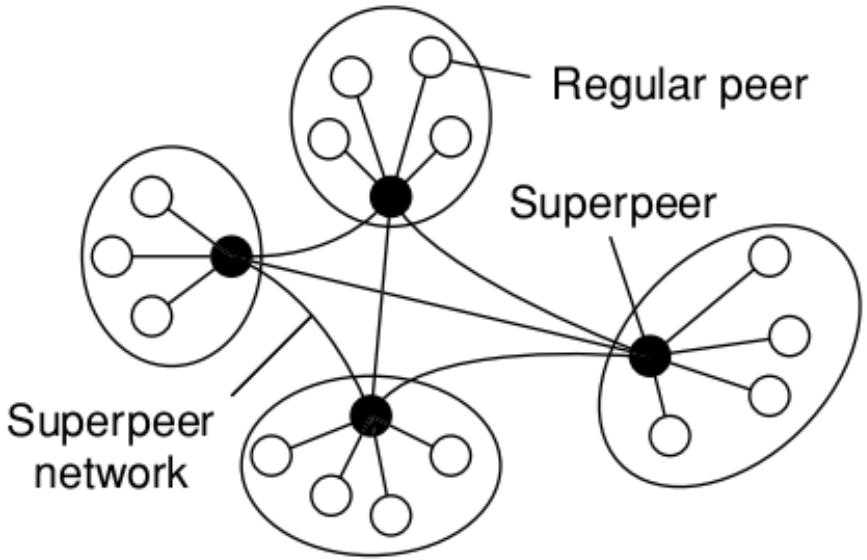
Unstructured P2P Networks

As the network grows, search time increases. (Can't deterministically look up the address of a node)

Can only flood the network with a search request.

Some networks have nodes that maintain an index of data items. Called **super peers**.

Requests are routed between super peers, to reduce flooding.



Hybrid Systems

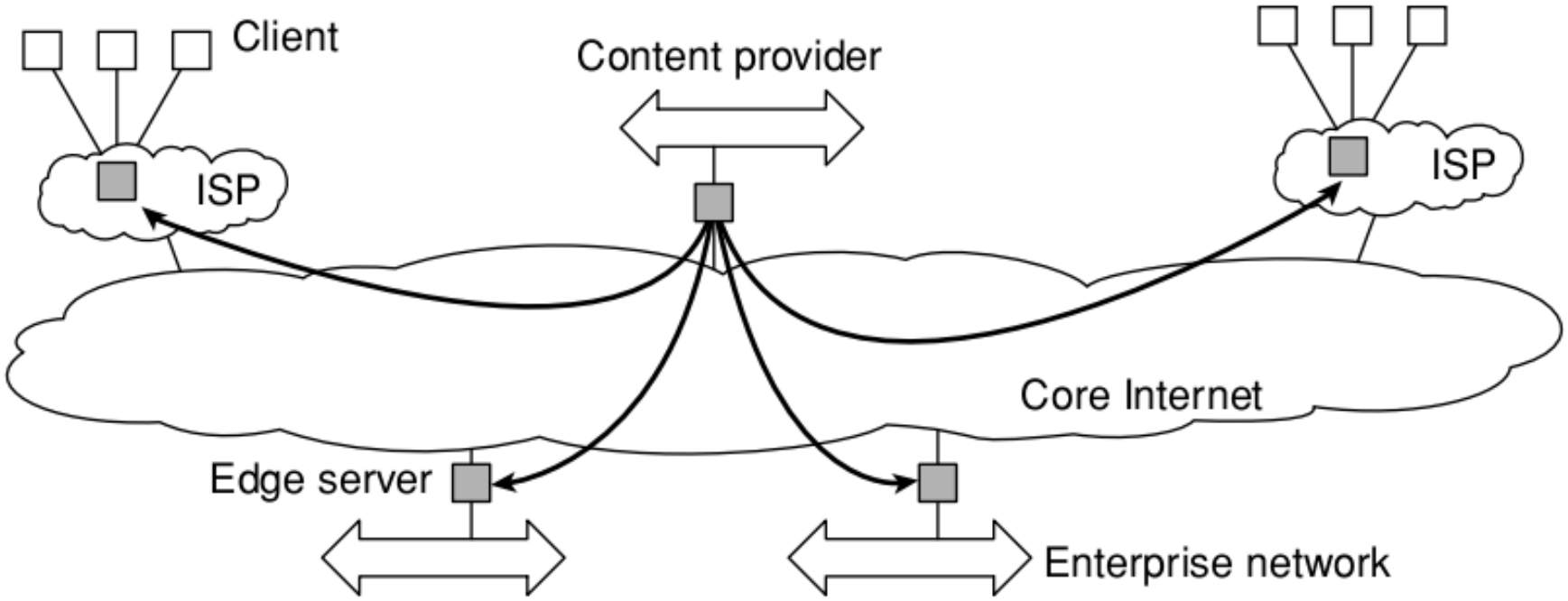
Hybrid Architectures

Combines features of **centralised** and **decentralised** systems.

An Edge-server system (ISP network) is a hybrid architecture.

The nodes in the private network are clients of the edge server.

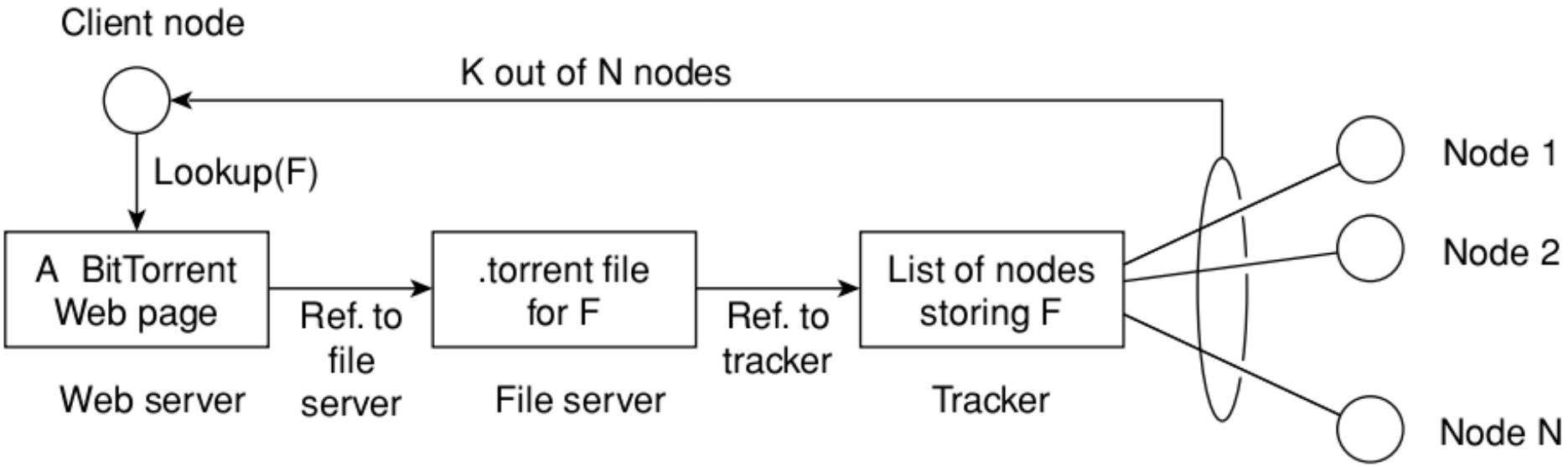
While the ISPs themselves provide translation between the private and public network



Hybrid Architectures - BitTorrent

Collaborative distributed systems. Initialisation is a problem.

Takes place through a normal client server architecture. Then the peer to peer network is then formed after the client gets a list of nodes to connect to on the network



Architectures vs Middleware

Architectures vs Middleware

Where does middleware fit in?

Ideally, middleware should provide **distribution transparency**. In practice, middleware will implement its own architectural style.

Advantage: If your application follows that style then programming on that middleware becomes simpler.

Disadvantage: this restricts the architectural styles available for your application, putting another style on top may reduce performance.

Wrappers

A special component that offers an interface acceptable to a client application.

Solves the problem of incompatible interfaces.

E.g. object adapter - component that allows applications to invoke remote objects

Important role in extending systems with existing components, and important for Openness

Can be done using brokers - e.g. message broker.

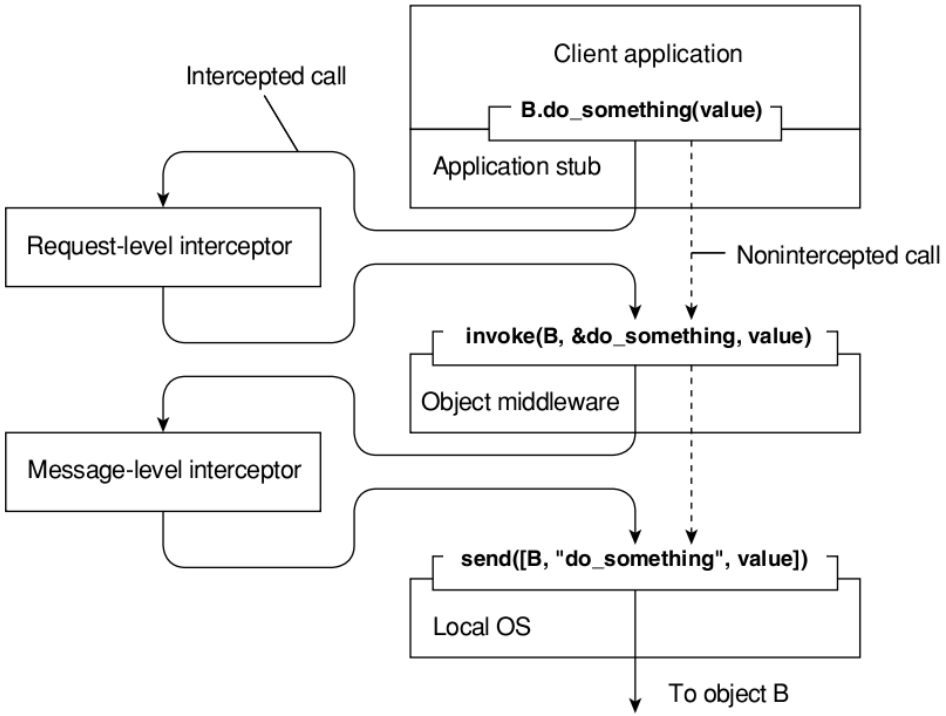
Interceptors

A software construct that breaks flow of control and permits other code to execute.

Object A can call a method that belongs to object B, while B is on a different node to A

Call will be intercepted by a **request level interceptor**.

Objects and calls may need to be transferred over the network. So a **message level interceptor** will be needed to send messages across the network



Adaptive Software

Rather than make the application respond to changes, the middleware should respond to changes.

For adaptive software to work it must implement:

- **Separation of concerns**: separate system components (functionality from reliability, security etc.)
- **Computational reflection**: allows the program to inspect itself and adapt (if necessary)
- **Component based design**: system is built from components at runtime/compile time.

Self-Management

Distributed systems become hard to manage as they increase in size and complexity.

Autonomic systems: a class of distributed systems that have feedback systems, permitting them to self control and adapt to changes.

i.e. self-managing, self-healing, self-configuring etc.

Self-Management

Feedback control loops are common to all autonomic systems.

Every component has controllable input parameters.

The system will also implement some form of monitoring. Large systems **provide estimates**.

Analysis components generate new inputs based on a **reference** and **measurements**.